XML Consortium

SOAの技術動向

XMLコンソーシアムSOA部会
日本IBM　天野富夫

XML Consortium

アジェンダ

- SOAとWebサービスのパターン
- Enterprise Integration PatternsとESB

XML Consortium

© XML Consortium

はじめに

- SOA(Service-Oriented Architecture)
  - システムを，業務視点の機能（サービス）の集合と捉え，ビジネス環境の変化に対して，迅速にサービスを組み換えることで柔軟に対応するシステム構築方針
- ESB(Enterprise Service Bus)
  - Webサービス、メッセージングミドルウェア、インテリジェントなルーティングと変換の機能を利用した新しいアーキテクチャ(Gartner 2002)

SOAとWebサービスのパターン

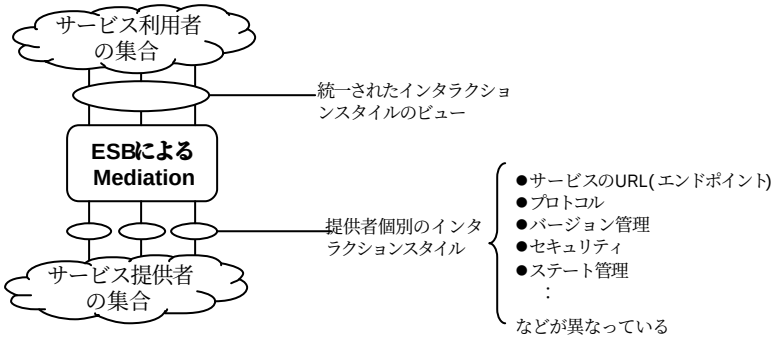
- パターンとは
  - システムを実現する上で繰り返し発生する要件や課題に対して解決策をまとめたもの
  - システムのどのレイヤーに注目するかによってさまざまな粒度のパターンが存在する
  - **あるパターンの実現に別のパターンが使われる**

ESBパターン

- 課題 :サービス同士をスムーズに連携させたい
  - サービス利用者と提供者の間の不整合/ギャップの解消
    - 個別の"利用者-提供者"の組ごとの実装は手間がかかる
  - ロケーションやメッセージフォーマットなど一方が変化したとき相手に影響を与えないようにする
- 解決 :サービス提供者と利用者を"バス"経由で接続する
  - ギャップの解消,ルーティング(1対1、1対多)の機能の機能をバスに集約する
  - プログラミングではなく設定変更で変化に対応する

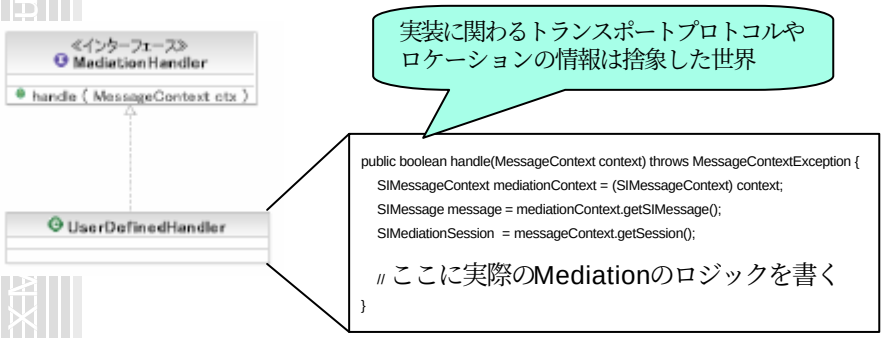
ESBパターンが提供するMediation

- サービスの使い勝手の統一
  - 利用者と提供者の間のギャップ/不整合の解消
  - 一方が変化したとき相手に影響を与えない
- プログラミングではなく設定変更で変化に対応する



ESBパターンの実現 :Mediationフレームワーク

- ESBはさまざまなMediationロジックを作るための フレームワークを提供する



Mediationの メリット

- サービスのロケーション(エンドポイント)の仮想化
  - サービスのアドレスはプログラムから切り離す
- 外部の設定情報によるルーティング(経路決定)
  - サービスの実ロケーションの情報を一括管理
- メッセージ内容によるルーティング
  - e.g メッセージフォーマットのバージョンをチェックして送 り先を変える
- メッセージ内容の変換サービス
- メッセージの分配(Disaggregation) と集約(Aggregation)
  - e.g 送 り先が1個増えたとき

XML Consortium

Enterprise Integration PatternsとESB

XML Consortium

Enterprise Integration Patterns

- G.Hohpe & B.Woolf, Enterprise Integration Patterns, Addison-Wesley
- 非同期メッセージングによるアプリケーション(サービス)統合のためのパターン集
  - Messaging Systems 6パターン
  - Messaging Channels 9パターン
  - Message Construction 9パターン
  - Message Routing 12パターン
  - Message Transfomation 6パターン
  - Messaging Endpoints 11パターン
  - System Management 8パターン

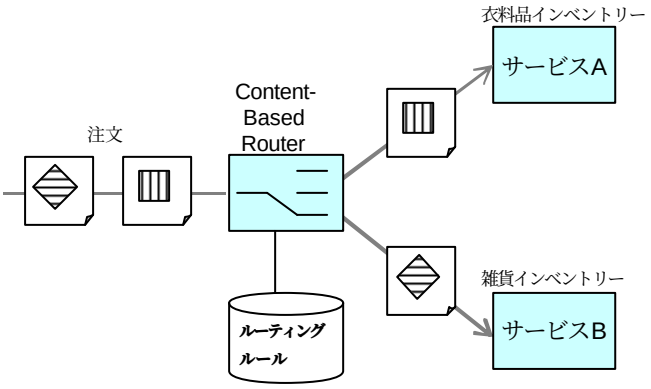
Why Enterprise Integration Patterns?

- SOAとはサービスの統合で物事を実現する
- サービスはメッセージのやりとりによって起動される
- メッセージのやりとりを柔軟に制御する必要がある
- メッセージのやりとりを柔軟かつ安全に設計するためのパターンがEnterprise Integration Patterns
- ESBはEnterprise Integration Patternsの特にMessage Routing関連のパターン(Patterns4e-businessのBroker/Routerパターン)の実現に役立つ

Message Routingパターン(1)

名称	課題
	解答
	SOA/Webサービスによる実現
Content-Based Router	論理的に単一のロジックの実装が複数の物理的システムに分散している状況をどう扱うか?
	Content-Based Routerが内容に応じてメッセージを正しい受信者(サービス)にルーティングする。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照
Message Filter	サービスが関係ないメッセージを受信するのを避けるには?
	Message Filterがある基準に従って関係の無いメッセージをチャネルから除去する。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照

Content-Based Routerパターン



Message Routingパターン(2)

名称	課題
	解答
	SOA/Webサービスによる実現
Dynamic Router	ルーティングの決定が宛先となるサービスの状態に依存している
	ルーティング対象のサービスから設定用メッセージを受信してルーティングルールを再構成する。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照 •ルーティング設定用Webサービスの提供
Recipient List	動的な受領者リストにあるサービスにのみメッセージをルーティングしたい
	受領者リストを生成またはメッセージから読みこんで対象者にのみルーティングする。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照

Message Routingパターン(3)

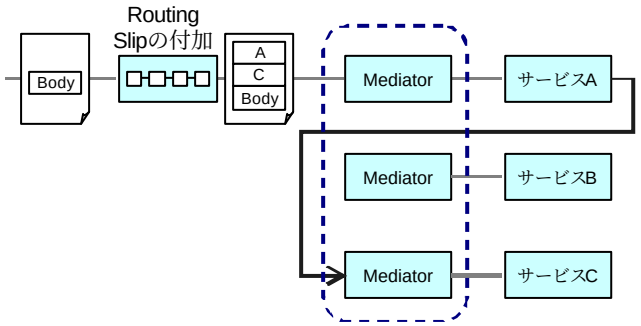
名称	課題
	解答
	SOA/Webサービスによる実現
Splitter	別々のやりかたで処理されるべき複数の要素を含んでいるメッセージをどのように扱うか？
	メッセージを個々の要素ごとに複数のメッセージに分割する。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照 •XSLT/DOM/SDO によるメッセージの生成
Aggregator	個別のしかし関連があるメッセージ群をまとめて処理するにはどうするか？
	ステートフルなフィルタを用意してメッセージ群を1つにまとめる。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照 •XSLT/DOM/SDO によるメッセージの生成

Message Routingパターン(4)

名称	課題
	解答
	SOA/Webサービスによる実現
Routing Slip	複数の処理過程を連続して通るメッセージの処理順が実行に一部変わる場合の対応
	処理順を記述したRouting Slipをメッセージに付与し、受信したサービスがSlipを参照して処理のスキップ等を行なう。
	•ESBのMediationフレームワークによる実装 •WS-RoutingによるRoting Slipの記述 •XSLT/DOMによるメッセージの変更 •XMLによるメッセージの記述/XPathによる参照
Process Manager	複数の処理過程が設計時に決まっておらずシーケンシャルでもない場合。
	次に処理すべきサービスを決定しプロセスの状態を管理するProcess Managerを導入する。
	•BPEL4WSエンジンによる実装

Routing Slipパターン

- 直列に実行される処理の順序を実行時の条件に応じて変更する

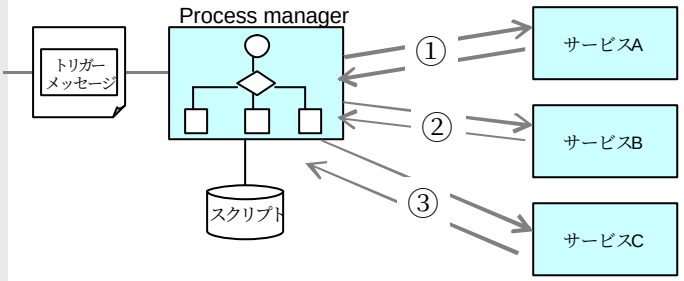


●Routing Slipの内容に従って経路を決定

●実体は一つで汎用的に使える

Process Managerパターン

- Process Manager(BPELエンジン)が決められたプロセス定義にしたがってサービスを呼出す
- 汎用的であり、他のRouting Patternの実装に使うこともできる



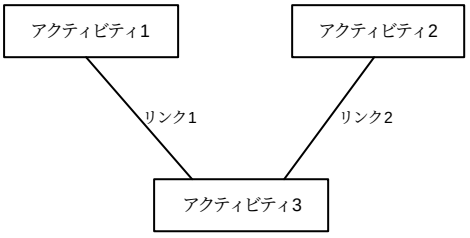
ワークフロー記述言語 :BPEL4WS

- Business Process Execution Language for Web Services
  - マイクロソフトのXLANGとIBMのWSFLを統合
  - OASISのWSBPEL TCで標準化活動中
  - PublicとPrivate両方のプロセスを実行可能

BPEL1.1の仕様はhttp://www-106.ibm.com/developerworks/webservices/library/ws-bpel/を参照

フロー制御のためのBPEL要素

- “アクティビティ”と”リング”による記述
  - アクティビティ：ワークフローにおける作業単位
  - リンク：アクティビティ間の順序関係をあらわす



①アクティビティの状態(正常終了/異常終了/実行対象にならない)が確定

②個々のリンクの評価値(正常終了ならtrue,それ以外はfalse)が定まる

③全ての入力リンクの値をJoin条件(and やorが指定できる)に従って評価。Ture ならばアクティビティを実行する

フロー制御のためのBPEL要素 (続き)

- アクティビティ
  - メッセージの送受信・操作に関わるアクティビティ要素
    - <receive>:要求の受信
    - <assign>:ワークフロー制御データ/関連データのセット
    - <invoke>:Webサービスの呼び出し
    - <reply>:応答の送信

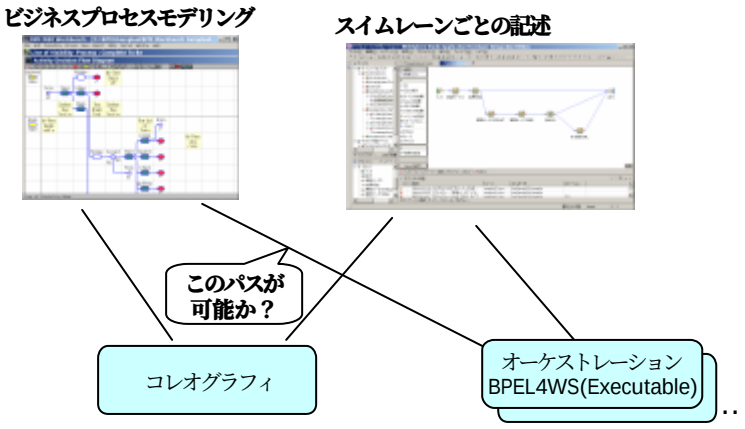
XML Consortium

フロー制御のためのBPEL要素 (続き)

- アクティビティ
  - アクティビティの構造化に関わるアクティビティ要素
    - <sequence>:複数のアクティビティを逐次結合
    - <switch>,<case>:条件分岐の制御構造
    - <pick>:非同期のイベントを記述
    - <while>:アクティビティの繰り返し構造
    - <flow>:アクティビティの並列実行
    - <scope>:障害処理や補償(compensation)範囲を指定

XML Consortium

プロセス定義



コレオグラフィとオーケストレーション

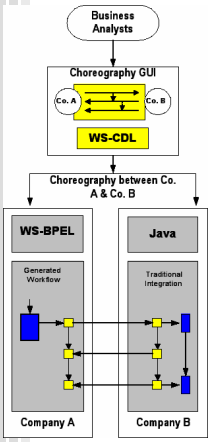
コレオグラフィの定義 (Web Services Glossary)

- 三人称
- A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.

オーケストレーションの定義 (Web Services Glossary)

- 一人称
- An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.

Web Services Choreography Description Language Version 1.0の仕様から



- コレオグラフィは企業間のサービスのインタラクションを規定し相互運用性を保証する
- 各企業内の実装はその企業の自由
  - 会社AはBPELを使用
  - 会社BはJ2EEで実装

BPEL4WSでコレオグラフィを記述できるという主張について

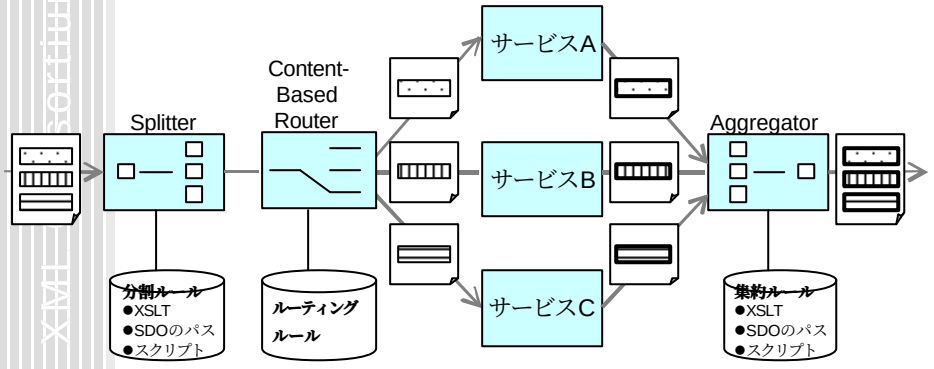
- BPELには2つの記述方式がある
  - 部分的に実装を不可視(opaque)にしたabstract process
  - 内部実装まで記述したexecutable process (オーケストレーション)
- コレオグラフィはAとBのインタラクションの記述
- abstract processでAはこうする、Bはこうする、という記述を行なえばコレオグラフィを記述したことになる、という主張と思われる。

Message Routingパターン(5)

名称	課題
	解答
	SOA/Webサービスによる実現
Composed Message Processor	メッセージが別々に処理されるべき複数の要素を持っているときの全体のメッセージの流れをどのように制御するか？
	メッセージを分割し、それぞれを適切な宛先にルーティングし、応答を1つにまとめる。
	•Splitter,Content-Based Router, Aggregatorの組合せによる。
Scatter-Gather	メッセージが複数のサービスに送られそれぞれが応答してくるとき全体のメッセージの流れをどのように制御するか？
	メッセージを複数のサービスにブロードキャストし応答を1つのメッセージに集約する。
	•Recipient ListまたはPublish-Subscribe ChannelとAggregatorの組合せによる。

Composed Message Processorパターン

- Splitter、Router、Aggregatorの組合せ



Message Routingパターン(6)

名称	課題
	解答
	SOA/Webサービスによる実現
Resequencer	関連するしかし順番の狂ったメッセージ群を正しく並べなおすには？
	ステートフルなフィルタを用意してメッセージを並べなおす。
	•ESBのMediationフレームワークによる実装 •XMLによるメッセージの記述/XPathによる参照
Message Broker	メッセージの宛先をサービスから分離し集中管理したい。
	メッセージの中継を行なうブローカーを用意する(Patterns4e-businessのブローカー)。
	•ESBの抽象化されたendpoint(destination)を用いて宛先を指定する。

Message Routing Patternが実現する疎結合性

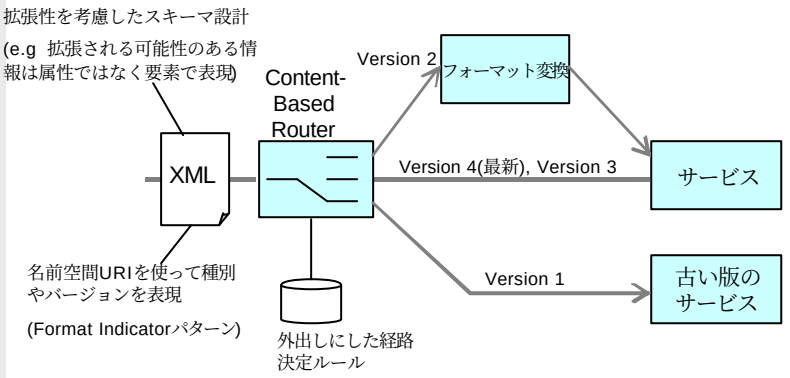
- 疎結合とは相互に作用しあうシステム間の依存性が最小であるような構成
  - ハードウェア/OSに関する依存性や実装プログラミング言語に関する依存性⇒SOAPやCorbaで最小化
  - ネットワーク上に分散しているオブジェクト間の依存性、使用するトランスポートプロトコルに関する依存性⇒SOAPで最小化
- 最小化できない依存性は外出しにして宣言的に扱えるようにする->コードへの影響を少なくする
  - 入出力メッセージフォーマットのWSDLによる記述
- Message Routing Patternはフローロジック(次にどのサービスを呼出すか)に関する依存性を外出しにすることで低減する1
  - Content-Based Router他あるいはProcess Manager
  - 手続き的(プログラム的)よりは宣言的な記述が望ましい

Message Transformationパターン(一部)

名称	課題
	解答
	SOA/Webサービスによる実現
Envelope Wrapper	ヘッダのフォーマットなどが決まっているメッセージングのやりとりに既存のシステムを参加させるには？
	既存システムのメッセージをエンベロープでラッピングして流す。
	•ラッパー/アンラッパーをESBのMediationフレームワークにより実装 •XMLによるメッセージの記述/名前空間を用いたボキャブラリの混在 •XPathによる参照/XSLTによるデータの抽出や変換
Cannonical Data Model	連携対象サービスが異なるメッセージフォーマットを採用しているときお互いの間の依存性を最小化するには？
	特定のサービスから独立したCannonical Data Modelに基づくフォーマットを設計し、個別フォーマットとの間で変換を行なう(変換はN2乗ではなくNのオーダーで済む)。
	•Cannonical Data Modelとの変換はESBのMediationフレームワークにより実装 •XSLTによる変換ルールの記述 •Cannonical Data Model 設計時のXMLスキーマのデザインパターンや共通部品(UBLなど)の利用

パターンの適切な組み合わせが変化への対応能力を実現する

■複数バージョンのサービスへの対応



Enterprise Integration PatternsとESB

- Enterprise Integration PatternsのMessage Routingパターン, Message Transformationパターンの多くはESBのMediationのフレームワークによって実現することができる
- その他XML/Webサービスの技術やベストプラクティスもEnterprise Integration Patternsの実現に役立つ